Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2010-027 May 14, 2010

Power-Aware Computing with Dynamic Knobs

Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Dynamic Knobs for Power-Aware Computing

Henry Hoffmann* Stelios Sidiroglou* Michael Carbin Sasa Misailovic
Anant Agarwal Martin Rinard
Computer Science and Artificial Intelligence Laboratory
{*hank,stelios,mcarbin,sasa,agarwal,rinard*}*@csail.mit.edu*

Abstract

We present PowerDial, a system for dynamically adapting application behavior to execute successfully in the face of load and power fluctuations. PowerDial transforms static configuration parameters into *dynamic knobs* that the PowerDial control system can manipulate to dynamically trade off the accuracy of the computation in return for reductions in the computational resources that the application requires to produce its results. These reductions translate into power savings.

Our experimental results show that PowerDial can enable our benchmark applications to execute responsively in the face of power caps (imposed, for example, in response to cooling system failures) that would otherwise significantly impair the delivered performance. They also show that PowerDial can reduce the number of machines required to meet peak load, in our experiments enabling up to a 75% reduction in direct power and capital costs.

1 Introduction

Energy efficiency is a first-order concern in the design of a range of computer systems. For mobile devices battery life is a primary issue. For embedded devices thermal constraints often limit peak power consumption. And power issues affect virtually every aspect of data center management, including operating expenses (e.g., server energy consumption), capital costs (e.g., power delivery and cooling infrastructure), environmental impact, and the need to deliver service through load and power fluctuations (from, for example, power caps imposed in response to events such as cooling system or primary power supply failure [1]).

A standard way to reduce power consumption is to reduce the delivered computational resources, typically by dynamic voltage and frequency scaling (DVFS), which can deliver cubic reductions in power consumption [34]. On modern computational platforms it is possible to switch (almost instantaneously) between a range of supported power states, each with different voltage, frequency, and power consumption characteristics [2]. Unfortunately, switching to a power state that reduces the delivered resources can impair or even disable the ability of applications to deliver acceptably responsive service. This issue is especially relevant for applications (such as video encoders or search engines) that need to provide fast interactive service to human users.

For many applications it is possible to reduce the accuracy (or quality) of the result in return for performance gains, which can then be translated into power savings. Many video encoders, for example, implement multiple encoding algorithms, with slower algorithms (which require more computational resources) producing higher-quality video and faster algorithms (which require fewer computational resources) producing lower-quality video [17]. Other examples include computations that perform heuristic searches of complex search spaces that contain multiple acceptable solutions (with some solutions potentially more desirable than others) [26, 21]. The flexibility to produce multiple answers allows system designers to incorporate an additional dimension into the power and performance optimization space: *accuracy*.

Applications often expose a static interface (in the form of configuration parameters) that enables users to control the application's performance versus quality of service (QoS) trade-off.[1] For example, x264 (a popular video encoder) provides command-line parameters that allow users to configure the motion estimation algorithm, the number of reference frames, and other aspects of the encoding algorithm. However, the need to execute successfully in environments in which the load or delivered computational resources may fluctuate makes static configuration problematic — terminating, then restarting applications with new configuration parameters may be undesirable or even unacceptable for long-running applications or applications with long restart times.

1.1 Basic Approach

We present a new system, PowerDial, for dynamically adapting the behavior of running applications to respond to fluctuations in load, power, or any other event that threatens the ability of the computing platform to deliver adequate computing power to satisfy demand:

*Stelios Sidiroglou and Henry Hoffmann have contributed equally to the research presented in this paper

[1] In this paper quality of service refers to the accuracy or quality of the result that the application produces, not the timing with which it produces or delivers this result.

- **Dynamic Knob Insertion:** PowerDial uses dynamic influence tracing to transform static application configuration parameters into dynamic control variables stored in the address space of the running application. These control variables are made available via a set of *dynamic knobs* that can change the configuration (and therefore the point in the trade-off space at which it executes) of a running application without interrupting service or otherwise perturbing the execution.
- **Dynamic Knob Calibration:** PowerDial explores the underlying performance versus QoS trade-off space (originally available via the configuration parameters) to characterize the performance and QoS of each dynamic knob setting.
- **Dynamic Knob Control:** PowerDial uses the Application Heartbeats framework [20] to dynamically monitor the performance of the application. When performance drops below target, the PowerDial control system uses the calibrated dynamic knobs to move the application to a more appropriate point in its trade-off space (the new point may, for example, give up some QoS in return for increased performance and decreased power consumption). PowerDial deploys a control algorithm and novel actuation strategy with provably good convergence and predictability properties.

1.2 Summary of Experimental Results

We evaluate the ability of PowerDial to control the behavior of four benchmark applications (the x264 video encoder, the bodytrack human body tracking application, the swaptions financial analysis application, and the swish++ search engine) to dynamically adapt to operate successfully in environments with fluctuating load and power characteristics. Our results show:

- **Trade-Off Space:** All of the applications exhibit a large viable trade-off space — three of the applications (x264, bodytrack, and swaptions) can execute from four to six times faster than their baseline (which maximizes QoS) with acceptable QoS losses. swish++ can execute approximately 1.5 times faster than its baseline (at the cost of dropping lower-ranked search results).
- **Power Capping:** PowerDial enables the applications to adapt effectively as a power cap (which reduces the processor frequency from 2.4 GHz to 1.6 Gz) is first imposed, then lifted. When the power cap is imposed, PowerDial preserves responsiveness by moving the applications to new Pareto-optimal points with more performance and slightly lower QoS. When the power cap is lifted, PowerDial restores the original QoS by moving the applications back to the baseline.
- **Peak Load Provisioning:** PowerDial makes it possible to service intermittent load spikes with fewer resources — when a load spike overwhelms the ability of the system to service the load with the baseline application configuration, PowerDial preserves responsive performance by dynamically reconfiguring the application to use less computation to produce (slightly) lower quality results. Because systems provisioned to meet peak loads with applications executing at baseline performance typically operate at low utilization levels [9], PowerDial makes it possible to significantly reduce the number of provisioned machines (by a factor of 3/4 for x264, bodytrack, and swaptions and a factor of 1/3 for swish++) required to service peak loads with acceptable QoS and (at most) negligible performance loss, all while providing baseline QoS for the vast majority of requests.

1.3 Contributions

This paper makes the following contributions:

- **Dynamic Knobs:** It introduces the concept of dynamic knobs, which manipulate control variables in the address space of a running application to dynamically change the point in the underlying performance versus QoS trade-off space at which the application executes.
- **PowerDial:** It presents PowerDial, a system that transforms static configuration parameters into calibrated dynamic knobs and uses the dynamic knobs to enable the application operate successfully in the face of fluctuating operating conditions (such as load spikes and power fluctuations).
- **Analysis and Instrumentation:** It presents the PowerDial analysis and instrumentation systems, which dynamically analyze the application to find and insert the dynamic knobs.
- **Control:** It presents the PowerDial control system, which uses a novel actuation algorithm and a control algorithm with provably good convergence and predictability properties to automatically optimize speed and accuracy to maintain desired performance while minimizing QoS loss.
- **Power Savings:** It shows how to use dynamic knobs to reduce the number of machines required to successfully service peak loads and to enable applications to tolerate the imposition of power caps. It shows how this ability can enable considerable power savings in a data center environment.
- **Experimental Results:** It presents experimental results that characterize the trade-off space that dynamic knobs make available in four benchmark applications. It also presents results that characterize the ability of PowerDial to enable applications to

dynamically adapt to fluctuating loads and power.

2 Dynamic Knobs

Dynamic knobs are designed for applications that 1) have command-line configuration parameters that control performance versus QoS trade offs and 2) use the Application Heartbeats API [20] (our system can automatically insert the required API calls, see Section 2.3). These applications typically exhibit the following general computational pattern:

- **Initialization:** During initialization the application parses and processes the configuration parameters and stores the resulting values in one or more control variables in the address space of the running application.
- **Main Control Loop:** The application executes multiple iterations of a main control loop. At each iteration it emits a heartbeat (by invoking the appropriate procedure in the Application Heartbeats API), reads the next unit of input, processes this unit, produces the corresponding output, then executes the next iteration of the loop. As it processes each input unit, it reads the control variables to determine which algorithm (or algorithm variant) to use.

With this computational pattern, the point in the performance versus QoS trade-off space at which the application executes is determined (by the configuration parameters) when the application starts and does not change during its execution. A goal of PowerDial, as illustrated in Figure 1, is to augment the application with the ability to dynamically change the point in the trade off space at which it is operating. At a high level, PowerDial accomplishes this goal as follows:

- **Parameter Identification:** The user of the program identifies a set of command-line configuration parameters and a range of settings for each such parameter. Each combination of parameter settings corresponds to a different point in the performance versus QoS trade-off space.
- **Dynamic Knob Identification:** For each combination of parameter settings, PowerDial uses dynamic influence tracing (which traces how the parameters influence values in the running application) to locate the control variables and record the values stored in each control variable.
- **Dynamic Knob Calibration:** Given a set of representative inputs and a QoS metric, PowerDial executes a training run for each input and combination of parameter settings. For each training run it records performance and QoS information. It then processes this information to identify the Pareto-optimal points in the explored performance versus QoS trade-off space.
- **Dynamic Knob Insertion:** PowerDial inserts callbacks that the PowerDial control system can use to set the control variables to values previously recorded during dynamic knob identification, thereby moving the application to a different Pareto-optimal point in the performance versus QoS trade-off space. Subsequent iterations of the main control loop will read the updated values in the control variables to (in effect) process further input as if the configuration parameters had been set to their corresponding different settings at application startup.

The result is an application that enables the PowerDial control system to dynamically control the point in the performance versus QoS trade-off space at which it executes. In standard usage scenarios PowerDial is given a target heart rate. If the application's dynamically observed heart rate is slower than the target, PowerDial uses the calibrated dynamic knobs to move the application to a point in the trade-off space with higher performance at the cost, typically small, of some QoS. If the observed heart rate is higher than the target, PowerDial moves the application to a point with lower performance and better QoS.

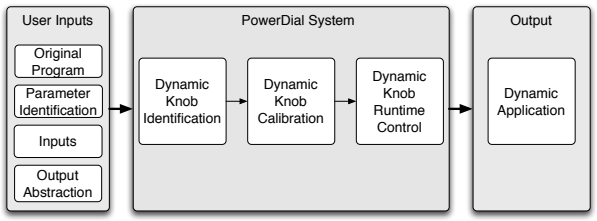


Figure 1: Dynamic Knob work flow.

2.1 Dynamic Knob Identification

For PowerDial to transform a given set of configuration parameters into dynamic knobs, it must identify a set of control variables that satisfy the following conditions:

- **Complete and Pure:** All variables whose values are derived from configuration parameters during application startup (before the application emits its first heartbeat) are control variables. The values of control variables are derived only from the given set of configuration parameters and not from other parameters.
- **Relevant and Constant:** During executions of the main control loop, the application reads but does not write the values of the control variables.

PowerDial uses influence tracing [11, 18] to find the control variables for the specified configuration parameters. For each combination of configuration parameter settings, PowerDial executes a version of the application

instrumented to trace, as the application executes, how the parameters influence the values that the application computes. It uses the trace information to find the control variables and record their values, applying the above conditions as follows:

- **Complete and Pure Check:** It finds all variables that, before the first heartbeat, contain values influenced by the specified configuration parameters. It checks that these values are influenced only by the specified configuration parameters.
- **Relevance Check:** It filters out any variables that the application does not read after the first heartbeat — the values of these variables are not relevant to the main control loop computation.
- **Constant Check:** It checks that the execution does not write a control variable after the first heartbeat.

Finally, PowerDial checks that the control variables are *consistent*, i.e., that the different combinations of parameter settings all produce the same set of control variables. If the application fails any of these checks, PowerDial rejects the transformation of the specified set of configuration parameters into dynamic knobs.

For each combination of parameter settings, PowerDial records the value of each control variable. The PowerDial control system uses this information to automatically change the values of the control variables at runtime. Note that because PowerDial uses a dynamic influence analysis to find the control variables, it is possible for unexercised execution paths to violate one or more of the above conditions. The influence analysis also does not trace indirect control-flow or array index influence. To enable a developer to (if desired) check that neither of these potential sources of imprecision affects the validity of the control variables, PowerDial produces a control variable report. This report lists the control variables, the corresponding configuration parameters from which their values are derived, and the statements in the application that access them. We have examined the reports for all of our benchmark applications (see Section 4) and verified that all of the automatically computed control variables are valid.

Our influence tracing system is implemented as a static, source-based instrumentor for C and C++. It is built on the LLVM compiler framework [11, 22] and inserts code to trace the flow of influence through the values that the application computes. For each value, it computes the command-line parameters that influenced that value. The currently implemented system supports control variables with datatypes of int, long, float, double, or STL vector. It augments the production version of the application with calls to the PowerDial control system to register the address of each control variable and read in the previously recorded values corresponding to the different dynamic knob settings. This mechanism gives the PowerDial control system the information it needs to apply a given dynamic knob setting.

2.2 Dynamic Knob Calibration

In this step, the PowerDial explores the performance versus QoS trade-off space available to the application via the specified configuration parameters. The user provides an application, a set of representative inputs, a set of specified configuration parameters (along with a range of values for each parameter), and a QoS metric. Given these values, PowerDial produces, for each combination of parameter settings, a specification of the point in the trade-off space to which the parameter settings take the application. This point is specified relative to the baseline performance and QoS of the parameter setting that delivers the highest QoS (which, for our set of benchmark applications, is the default parameter setting).

The PowerDial calibrator executes all combinations of the representative inputs and configuration parameters. For each parameter combination it records the mean (over all representative inputs) speedup of the application. It computes the speedup as the execution time of the application running with the default parameter settings divided by the execution time of the application with the current parameter combination. In a separate instrumented execution, it also records the values of the control variables (see Section 2.1).

For each combination of configuration parameters PowerDial also records the mean (over all representative inputs) QoS. The QoS metric works with a user-provided, application-specific *output abstraction* which, when provided with an output from the program, produces a set of numbers $o_1, \ldots, o_m$. The output abstraction typically extracts relevant numbers from the output or computes a measure of output quality (such as, for example, the peak signal-to-noise ratio of the output). Given the output abstraction from the baseline execution $o_1, \ldots, o_m$ and an output abstraction $\hat{o}_1, \ldots, \hat{o}_m$ from the execution with the current parameter settings, we compute the QoS as the *distortion* [28]:

$$qos = \frac{1}{m} \sum_{i=1}^{m} w_i \left| \frac{o_i - \hat{o}_i}{o_i} \right| \quad (1)$$

Here each weight w_i is optionally provided by the user to capture the relative importance of the ith component of the output abstraction. Note that a qos of zero indicates optimal QoS, with higher numbers corresponding to worse QoS. PowerDial supports caps on QoS loss — if a specific parameter setting produces a QoS loss exceeding a user-specified bound, the system can exclude the corresponding dynamic knob setting from further consideration.

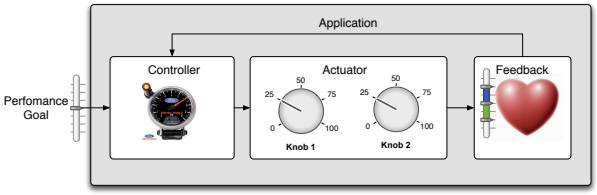


Figure 2: The PowerDial Control System.

2.3 The PowerDial Control System

The PowerDial control system automatically adjusts the dynamic knobs to appropriately control the application. As shown in Figure 2, the PowerDial control system contains the following components: a feedback mechanism that allows the system to monitor the performance of the application, a control component which converts the feedback into a desired speedup, and an actuator which converts the desired speedup into settings for one or more dynamic knobs.

Feedback Mechanism PowerDial uses the Application Heartbeats framework as its feedback mechanism [20]. In general, PowerDial can work with any application that has been engineered to use this framework to emit heartbeats at regular intervals and express a desired performance in terms of a target minimum and maximum heart rate. For our set of benchmark applications, the PowerDial instrumentation system automatically inserts the API calls that emit heartbeats — it profiles each application to find the most time-consuming loop (in all of our applications this is the main control loop), then inserts a heartbeat call at the top of this loop. In general, the PowerDial control system is designed to work with any target minimum and maximum heart rate that the application can achieve. For our experiments (see Section 4), the minimum and maximum heart rate are both set to the average heart rate measured for the application using the default command line.

Control Strategy PowerDial employs a generalized control strategy which can be used to control any application which uses Application Heartbeats as a feedback mechanism [23]. This controller monitors the feedback mechanism and determines both when to speedup or slowdown the application as well as how much speedup or slowdown to apply. To use this control strategy, PowerDial must work with a model of the application's *workload* or the average amount of time between two heartbeats. PowerDial obtains this model by first computing the average heart rate over all the representative inputs and then inverting that number. If the training inputs provided by the user are representative of the production inputs, this workload will be accurate and the control strategy can guarantee several desirable properties. First, it is stable, i.e., the performance of the application will not oscillate between high and low values. Second, it is accurate, i.e., the application will reach the desired performance. Third, the controller has bounded settling time, i.e., the application will reach the desired performance in finite time. Finally, the system has a known overshoot and thus the difference between the maximum speed of the application and the desired speed is bounded.

Actuation Policy The PowerDial actuator must convert the speedup specified by the controller into a dynamic knob setting. The controller is a continuous linear system, and thus, the actuator must convert the continuous signal into actions that can be realized in the application's discrete, potentially non-linear dynamic knob system. For example, the controller might specify a speedup of 1.5 while the smallest speedup available through a knob setting is 2. To resolve this issue, the actuator computes a set of actions to take over a time quantum heuristically determined to be the time required to process twenty heartbeats. In this example, the actuator would run with a speedup of 2 for half the time quantum and the default speedup of 1 for the other half.

In the general case, the actuator determines which actions to take for the next time quantum by optimizing a system of linear constraints. Let H_o be the observed heart rate, while H_t is the target heart rate of the system. Let S_{max} be the maximum achievable speedup for the application given its dynamic knobs, and let S_{min} be the minimum speedup corresponding to a knob setting such that $S_{min} \geq H_t/H_o$. Let unknowns t_{max}, t_{min}, and $t_{default}$ correspond to the percentage of time during the next quantum to run with the application's knobs set to the maximum speedup, the minimum required speedup, and the default settings, respectively. Then, the following system of constraints captures the behaviors the actuator considers for the next time quantum.

$$S_{max} \cdot t_{max} + S_{min} \cdot t_{min} + \frac{H_o}{H_t} \cdot t_{default} = 1 \quad (2)$$

$$t_{max} + t_{min} + t_{default} \leq 1 \quad (3)$$

$$t_{max}, t_{min}, t_{default} \geq 0 \quad (4)$$

While there are many solutions to this system of constraints, two are of particular interest for making power/performance/QoS trade offs. First, for platforms with sufficiently low idle power consumption (see Section 3), PowerDial sets $t_{min} = t_{default} = 0$, forcing the application to run at the highest available speedup. If $t_{max} < 1$ the system can idle for the remaining $1 - t_{max}$ portion of the time quantum to save power. The second solution PowerDial considers results from setting $t_{max} = 0$ and requiring $t_{min} + t_{default} = 1$. This solution will run the application at the lowest obtainable speedup that will enable the application to meet its heart rate target. This solution therefore delivers the lowest feasible QoS loss.

Having determined values for t_{max}, t_{min}, and $t_{default}$ for the next time quantum, the PowerDial controller executes the corresponding plan, then computes a new plan when the quantum expires.

3 Saving power using Dynamic Knobs

In this section, we describe how dynamic knobs can enable power savings in large scale data centers by:

- **DVFS:** Enabling DVFS savings for applications with responsiveness requirements.
- **Elastic Response:** Enabling *elastic response*, a technique that makes applications able to operate successfully in the presence of environmental fluctuations (caused, for example, by phenomena such as power caps and load spikes).
- **Consolidation:** Introducing an alternative server consolidation technique that enables a system to acceptably process peak loads with fewer machines.

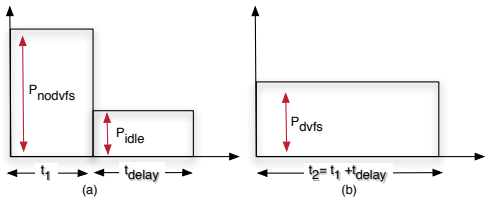


Figure 3: DVFS Energy Savings

DVFS Figure 3 shows how operating at lower power states can enable systems to reduce power consumption at the cost of increased latency. The area within the boxes represents the total energy required to complete a workload. For a task which takes time t and consumes average power of P_{avg}, the total energy can be calculated as: $E_{task} = P_{avg} \cdot t$. Without DVFS (Figure 3 (a)), the workload consumes power P_{nodvfs} for time t_1 and power P_{idle} for the remaining time t_{delay}. With DVFS (Figure 3 (b)), the consumed power is reduced to P_{dvfs} but the execution time increases to $t_2 = t_1 + t_{delay}$. To accurately calculate DVFS energy savings, the idle power consumed by the non-DVFS system (P_{idle}) must be included. Thus the energy savings due to DVFS can be computed as:

$$E_{dvfs} = (P_{nodvfs} \cdot t_1 + P_{idle} \cdot t_{delay}) - (P_{dvfs} \cdot t_2) \quad (5)$$

For CPU-bound applications, t_2 can be predicted by the change in operating frequency as: $t_2 = \frac{f_{nodvfs}}{f_{dvfs}} \cdot t_1$. We note that any power savings here come at the cost of added latency.

Dynamic knobs can complement DVFS by allowing systems to save power by reducing the amount of computational resources required to accomplish a given task. There are two cases to consider depending on the idle power of the system P_{idle} as illustrated in Figure 4. Figure 4(a) illustrates the first case. This case applies to systems with low idle power consumption (i.e., small P_{idle}), a common case in mobile devices. In this case, the best energy savings strategy is to complete the task as quickly as possible, then return to the low-power idle state, a strategy known as *race-to-idle*. Dynamic knobs can facilitate race-to-idle operation by decreasing the amount of computational resources required to complete the task (in return for some QoS loss), thereby reducing t_1. Figure 4(b) illustrates the second case, which applies to systems with high idle power consumption (i.e., large P_{idle}), common in current server class machines. In this case, dynamic knobs can allow the system to operate at a lower power state for the time t_2 allocated to complete the task.

In both cases the energy savings available through combining DVFS and dynamic knobs can be calculated as:

$$E_{savings} = (P_{nodvfs} \cdot \frac{t_1}{S(QoS)}) + P_{idle} \cdot t_{delay}) - (P_{dvfs} \cdot t_2) \quad (6)$$

where $S(QoS)$ represents the speedup available as a function of acceptable QoS loss (i.e., the desired level of accuracy).

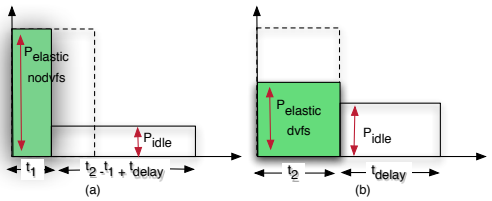


Figure 4: Energy Savings with DVFS and Dynamic Knobs

Elastic response Dynamic knobs enhance systems with an elastic response mechanism that can dampen the effects of several power saving techniques (or any other event that affects the delivered computational resources) on latency. Elastic response enables applications to dynamically degrade the QoS they deliver in return for performance improvements. This ability can enable techniques, such as power capping and server consolidation, without increasing application latency.

Server consolidation Dynamic knobs can enable power savings in the data center by reducing the total number of machines required to meet peak load without increasing latency. This is achieved by increasing the throughput of each individual machine (trading QoS for performance) when the system experiences load spikes that exceed its capacity to service when operating with the default application configuration. The resulting reduction in computational resources required to service the load enables the system to service the load spike without increasing the service time. The reduction in the number of machines improves server utilization during normal operation and reduces energy wasted on idle re-

sources. It also provides extra savings in the form of reduced cooling costs.

To quantify how dynamic knobs can help improve server consolidation, we need to examine the total work required to meet system peak load requirements. This can be calculated as follows:

$$W_{total} = (W_{machine} \cdot N_{orig}) \tag{7}$$

W_{total} represents the total work done by the data center, where $W_{machine}$ represents the work done per machine and N_{orig} is the total number of machines. Let $S(QoS)$ be the speedup achieved as a function of QoS degradation. For maximum quality of service the speedup is 1, that of the original system. The number of machines required to meet peak load with some loss of accuracy can be shown as:

$$N_{new} = \frac{W_{total}}{S(QoS)} \cdot \frac{1}{W_{machine}} \tag{8}$$

Here N_{new} is the new, lower number of machines that can be used to meet peak load requirements after consolidation. To measure the savings achieved by this consolidation let $U_{new} = N_{orig}/\lceil N_{new} \rceil$ be the average utilization of the consolidated system and U_{orig} be the average utilization in the original system. Further assume that the two systems both use machines that consume power P_{load} under load and P_{idle} while idle. Let P_{orig} be the average power in the original system, while P_{new} is the average power in the smaller consolidated system. Then we can calculate the average power savings of the consolidated system as:

$$P_{orig} = N_{orig}(U_{orig} \cdot P_{load} + (1 - U_{orig})P_{idle}) \tag{9}$$

$$P_{new} = N_{new}(U_{new} \cdot P_{load} + (1 - U_{new})P_{idle}) \tag{10}$$

$$P_{save} = P_{orig} - P_{new} \tag{11}$$

In existing data centers, this power savings can reduce both direct costs (e.g., the energy bill) and indirect costs (e.g., cooling and conversion costs). The enabled consolidation can also reduce the capital costs of new data centers. Considering that electricity costs in the U.S. are approximately $0.8/Watt-year (excluding cooling and conversion costs, which typically add a 50% energy overhead) and that capital costs for typical Tier-2 datacenters are in the range of $10 and $20 per deployed Watt [14], dynamic-knob-enabled server consolidation can deliver significant cost savings.

4 Benchmarks and Inputs

We report results for four benchmarks. Swaptions, bodytrack, and x264 are all taken from the PARSEC benchmark suite [10]; swish++ is as open-source search engine [32]. For each application we acquire a set of representative inputs, then randomly partition the inputs into *training* and *production* sets. We use the training inputs to obtain the dynamic knob response model (see Section 2) and the production inputs to evaluate the behavior on previously unseen inputs. Table 1 summarizes the sources of these inputs. All of the applications support both single- and multi-threaded execution. In our experiments we use whichever mode is appropriate.

4.1 swaptions

Description: This financial analysis application uses Monte Carlo simulation to solve a partial differential equation that prices a portfolio of swaptions.

Knobs: We use a single command line parameter, `-sm` as the dynamic knob. This integer parameter controls the number of Monte Carlo simulations for each swaption. The values range from 10,000 to 1,000,000 in increments of 10,000; one million is the default value for the PARSEC native input.

Inputs: Each input contains a set of parameters for a given swaption. The native PARSEC input simply repeats the same parameters multiple times, causing the application to recalculate the same swaption price. We augment the evaluation input set with additional randomly generated parameters so that the application computes prices for a range of swaptions.

QoS Metric: Swaptions prints the computed prices for each swaption. The QoS metric computes the distortion of the swaption prices (see Equation 1), weighting the prices equally. This metric directly captures the ability of the application to produce accurate swaption prices.

4.2 x264

Description: This media application encodes a raw (uncompressed) video according to the H.264 standard [36].

Knobs: We use three knobs: `--subme` (an integer parameter which determines the algorithms used for subpixel motion estimation), `--merange` (an integer which governs the maximum search range for motion estimation), and `--ref` (which specifies the number of reference frames searched during motion estimation). `--subme` ranges from 1 to 7, `--merange` ranges from 1 to 16, and `--ref` ranges from 1 to 5. In all cases higher numbers correspond to higher quality encoded video and longer encoding times. The PARSEC defaults for these are 7, 16, and 5, respectively.

Inputs: The native PARSEC input contains a single high-definition (1080p) video so we use additional 1080p inputs from xiph.org [5].

QoS Metric: The QoS metric is the distortion of the peak signal to noise ratio (PSNR, as measured by the H.264 reference decoder [19]) and bitrate (as measured by the size of the encoded video file), with the PSNR and bitrate weighted equally. This QoS metric captures the two most important attributes of encoded video: image quality and compression.

Benchmark	Training Inputs	Production Inputs	Source
swaptions	64 swaptions	512 swaptions	PARSEC & randomly generated swaptions
x264	4 HD videos of 200+ frames	12 HD videos of 200+ frames	PARSEC & xiph.org [5]
bodytrack	sequence of 100 frames	sequence of 261 frames	PARSEC & additional input by authors
swish++	2000 books	2000 books	Project Gutenberg [3]

Table 1: Summary of Training and Production Inputs for Each Benchmark

4.3 bodytrack

Description: This computer vision application uses an annealed particle filter and videos from multiple cameras to track a human's movement through a scene [13].

Knobs: bodytrack uses positional parameters, two of which we convert to knobs: `argv[5]`, which controls the number of annealing layers, and `argv[4]`, which controls the number of particles. The number of layers ranges from 1 to 5 (the PARSEC default); the number of particles ranges from 100 to 4000 (the PARSEC default) in increments of 100.

Inputs: bodytrack requires data collected from four carefully calibrated cameras. We use of sequence of 100 frames (obtained from the maintainers of PARSEC) as the training input and the PARSEC native input (a sequence of 261 frames) as the production input.

QoS Metric: bodytrack produces two outputs: a text file containing a series of vectors representing the positions of the body over time and a series of images graphically depicting the information in the vectors overlaid on the video frames from the cameras. The QoS metric is the distortion of the vectors that represent the body position. The weight of each vector is proportional to its magnitude. Vector components which represent larger body parts (such as the torso) therefore have a larger influence on the QoS metric than vectors that represent smaller body parts (such as forearms).

4.4 swish++

Description: This search engine is used to index and search files on web sites. Given a query, it searches its index for documents that match the query and returns the documents in rank order. We configure this benchmark to run as a server — all queries originate from a remote location and search results must be returned to the appropriate location.

Knobs: We use the command line parameter `--max-results` (or `-m`, which controls the maximum number of returned search results) as the single dynamic knob. We use the values 5, 10, 25, 50, 75, and 100 (the default value).

QoS Metric: We use F-measure [24] (a standard information retrieval metric) as our QoS metric. F-measure is the harmonic mean of the *precision* and *recall*. Given a query, precision is the number of returned documents that are relevant to the query divided by the total number of returned documents. Recall is the number of relevant returned documents divided by the total number of relevant documents (returned or not). We examine precision and recall at different cutoff values, using typical notation **P @N**.

Inputs: We use public domain books from Project Gutenberg [3] as our search documents. We use the methodology described by Middleton and Baeza-Yates [25] to generate queries for this corpus. Specifically, we construct a dictionary of all words present in the documents, excluding stop words, and select words at random following a power law distribution. We divide the documents randomly in to equally-sized training and production sets.

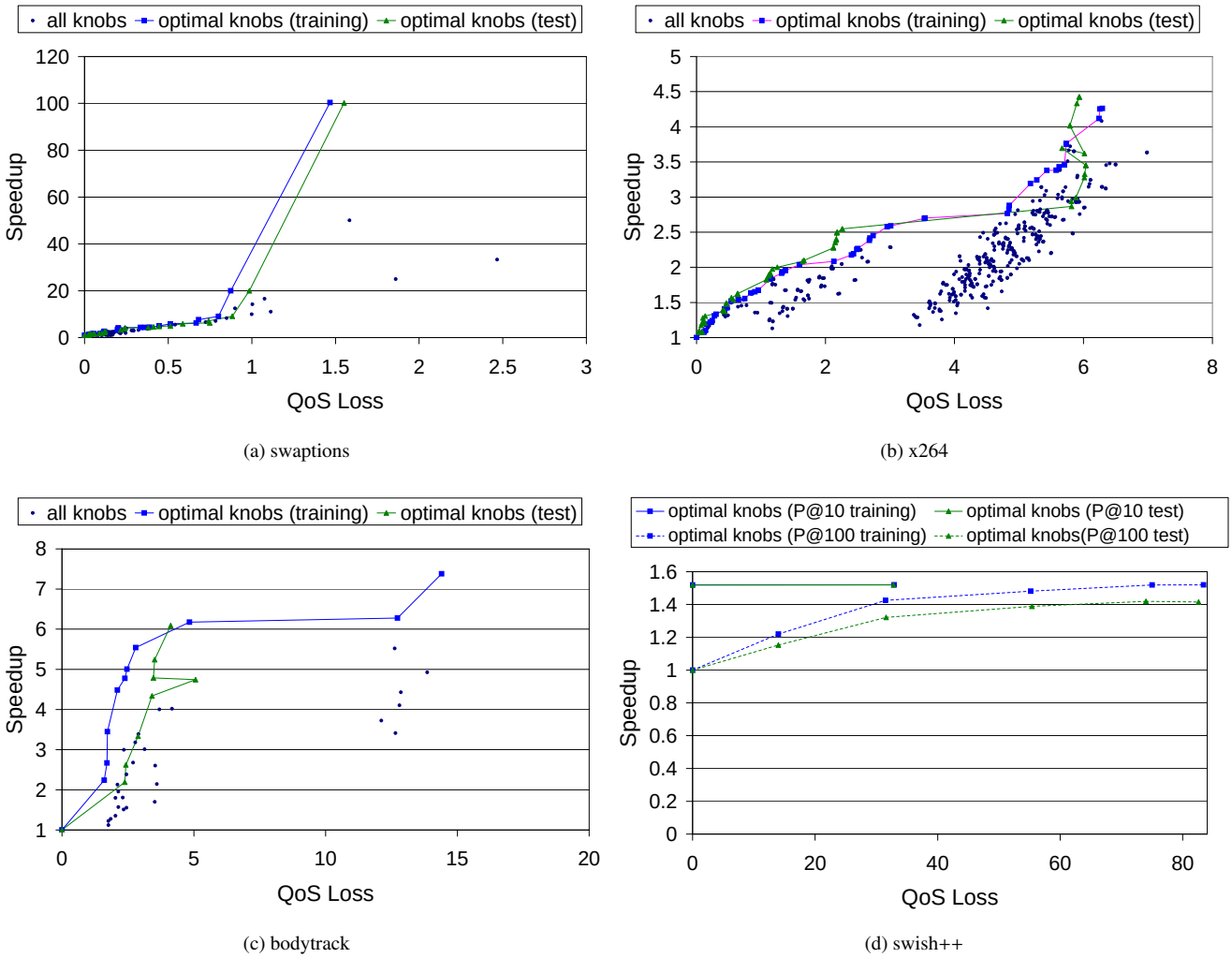
5 Experimental Evaluation

In this section, we discuss the experiments used to evaluate the PowerDial system. We first describe our experimental platform, then each of the experiments conducted. We begin by testing how well speedup and QoS measured on training inputs correlate with that measured on the production inputs. We next demonstrate the use of dynamic knobs to maintain performance while trading some small QoS loss for significant power savings. Then, we show how dynamic knobs can be used to maintain performance in response to external events such as the imposition of a power-cap. Finally, we show how dynamic knobs can be used to maintain performance while reducing the number of machines in the system, saving both power and infrastructure cost.

5.1 Experimental Platform

We run all our experiments on a Dell PowerEdge R410 server with two quad-core Intel Xeon E5530 processors running Linux 2.6.26. The processors support seven power states with clock frequencies from 2.4 GHz to 1.6 GHz. The `cpufrequtils` package enables software control of the clock frequency (and thus the power state). We use a WattsUp device to sample and store the consumed power at 1 second intervals [4]. All benchmark applications run for significantly more than 1 second. The maximum and minimum measured power ranges from 220 watts (at full load) to 80 watts (idle), with a typical idle power consumption of approximately 90 watts.

We measure the overhead of the PowerDial control system by comparing the performance of the benchmarks with and without the control system. The overhead of the PowerDial control system is insignificant and within the run-to-run variations in the execution times of the benchmarks executing without the control system.



(a) swaptions

(b) x264

(c) bodytrack

(d) swish++

Figure 5: QoS loss versus speedup for the three PARSEC benchmarks.

5.2 Performance/QoS Trade-Offs

Dynamic knobs modulate power consumption by controlling the amount of computational work required to perform a given task. On a machine that delivers constant baseline performance (i.e., no clock frequency changes), changes in computational work correspond to changes in execution time.

Figures 5a–5d present the points that dynamic knobs make available in the speedup versus QoS trade-off space for each benchmark application. The points in the graphs plot the observed mean (across the training or production inputs as indicated) speedup as a function of the observed mean QoS loss for each dynamic knob setting. Blue points plot results for the training inputs, with blue squares (connected by a line) indicating Pareto-optimal dynamic knob settings. Green triangles (again connected by a line) plot the corresponding points for these Pareto-optimal dynamic knob settings for the production inputs. All speedups and QoS losses are calculated relative to the dynamic knob setting which delivers the highest QoS (and consequently the largest execution time). We observe the following facts:

- **Effective Trade-Offs:** Dynamic knobs provide access to operating points across a broad range of speedups (up to 100 for swaptions, 4.5 for x264, and 7 for bodytrack). Moreover, QoS losses are acceptably small for virtually all Pareto-optimal knob settings (up to only 1.5% for swaptions, 7% for x264, and, for speedups up to 6, 6% for bodytrack).
  For swish++, dynamic knobs enable a speedup of up to approximately a factor of 1.5. The QoS loss increases linearly with the dynamic knob setting. The effect of the dynamic knob is, however, very simple — it simply drops lower-priority search results. So, for example, at the top dynamic knob setting, swish++ returns the top five search results.
- **Close Correlation:** To compute how closely behavior on production inputs tracks behavior on training inputs, we take each metric (speedup and QoS loss), compute a linear least squares fit of training data to

Benchmark	Speedup	QoS Loss
x264	0.995	0.975
bodytrack	0.999	0.839
swaptions	1.000	0.999
swish++	0.996	0.999

Table 2: Correlation coefficient of observed values from training with measured values on test inputs.

production data, and compute the correlation coefficient of each fit (see Table 2). The correlation coefficients are all close to 1, indicating that behavior on training inputs is an excellent predictor of behavior on production inputs.

5.3 Power/QoS Tradeoffs

To characterize the power versus QoS trade-off space that dynamic knobs make available, we initially configure each application to run at its highest QoS point on a processor in its highest power state (2.4 GHz) and observe the performance (mean time between heartbeats). We then instruct the PowerDial control system to maintain the observed performance, use `cpufrequtils` to drop the clock frequency to each of the six lower-power states, run each application on all of the production inputs, and measure the resulting performance, QoS, and mean power consumption (the mean of the power samples over the execution of the application in the corresponding power state). We verify that, for all power states, PowerDial delivers performance within 5% of the target.

Figures 6a–6d plot the resulting QoS (right y axis, in percentages) and mean power (left y axis) as a function of the processor power state. For x264, the combination of dynamic knobs and frequency scaling can reduce system power by as much as 21% for less than 0.5% QoS loss. For bodytrack, we observe a 17% reduction in system power for less than 2.3% QoS loss. For swaptions, we observe an 18% reduction in system power for less than .05% QoS loss. Finally, for swish++ we observe power reductions of up to 16% for under 32% QoS loss. Note that for swish++ all of the QoS loss comes from reduced recall. The dynamic knob simply truncates the list of returned results — the top results are the same, but swish++ returns fewer total results.

5.4 Elastic Response to Power Capping

The PowerDial system makes it possible to dynamically adapt application behavior to preserve performance (measured in heartbeats) in the face of *any* event that degrades the computational capacity of the underlying platform. We next investigate a specific scenario — the external imposition of a temporary power cap via a forced reduction in clock frequency. We first start the application running on a system with uncapped power in its highest power state (2.4 GHz). We instruct the PowerDial control system to maintain the observed performance (time between heartbeats). Approximately one quarter of the way through the computation we impose a power cap that drops the machine into its lowest power state (1.6 GHz). Approximately three quarters of the way through the computation we lift the power cap and place the system back into its highest power state (2.4 GHz).

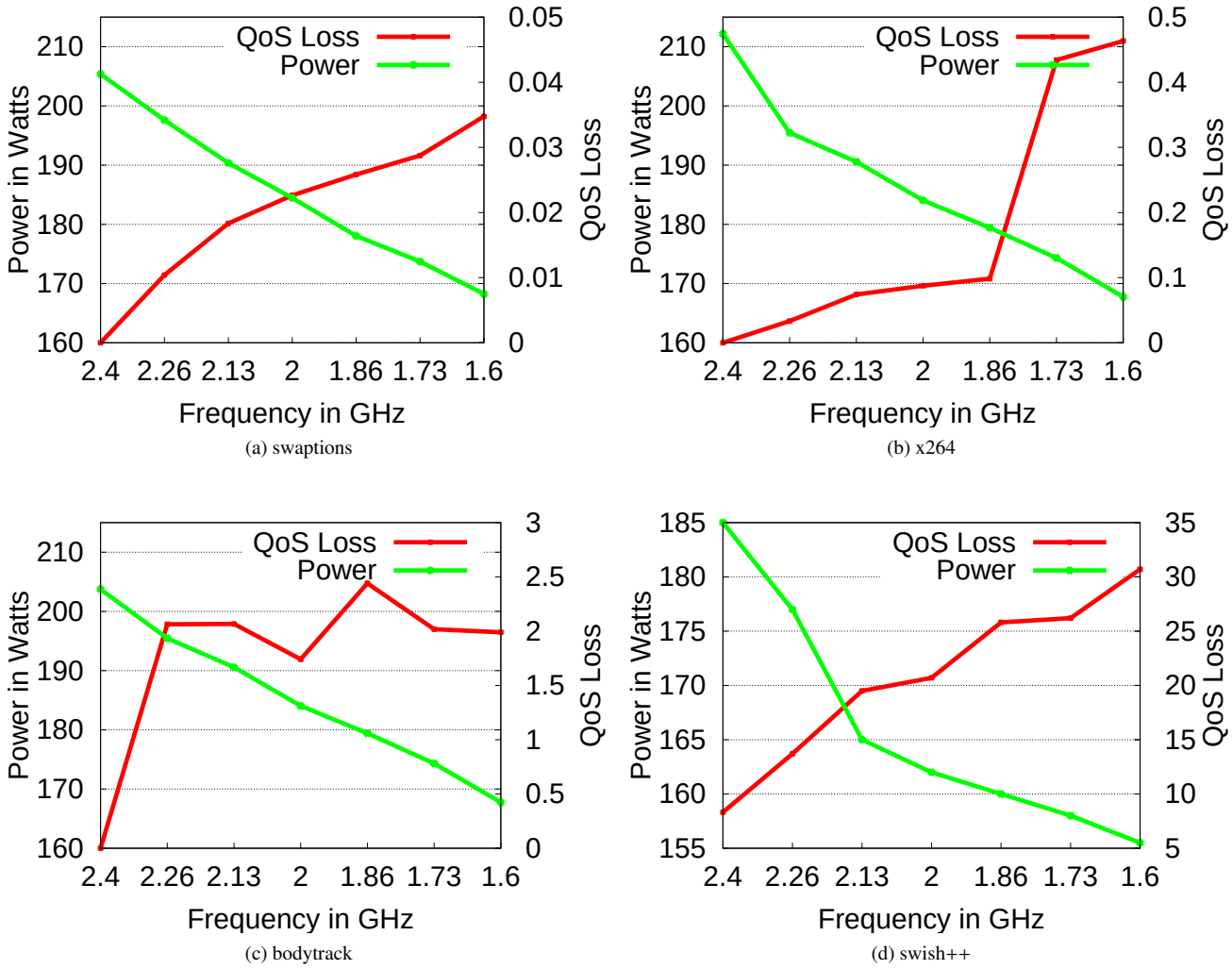
Figures 7a–7d present the dynamic behavior of the benchmarks as they respond to the power cap and corresponding processor frequency changes. Each graph plots the observed performance (computed as the sliding mean of the last twenty times between heartbeats times normalized to the target heart rate of the application) of the application (left y axis) as a function of time. We present the performance of three versions of the application: a version without dynamic knobs (red points), a baseline version running with no power cap in place (blue points), and a version that uses dynamic knobs to preserve the performance despite the power cap (green points). We also present the knob "gain" or the instantaneous speedup achieved by the dynamic knob runtime (right y axis).

All applications exhibit the same general pattern. At the imposition of the power cap, the dynamic knob adjustments kick in, the gain increases (red Knob Gain line), and the performance of the application first spikes down (green points), then returns back up to the baseline performance. When the power cap is lifted, the dynamic knobs adjust again, the gain decreases, and the application performance returns to the baseline after a brief upward spike. For most of the first and last quarters of the execution, the application executes with essentially no QoS loss. For the middle half of the execution, the application converges to the low power operating point plotted in Figures 6a–6d as a function of the 1.6 GHz processor frequency. Without dynamic knobs (red points), application performance drops well below the baseline as soon as the power cap is imposed, then rises back up to the baseline only after the power cap is lifted.

Within this general pattern the applications exhibit varying degrees of noise in their response. Swaptions exhibits very predictable performance over time with little noise. swish++, on the other extreme, has relatively unpredictable performance over time with significant noise. x264 and bodytrack fall somewhere in between. Despite the differences in application characteristics, our dynamic adaptation mechanism makes it possible for the applications to largely satisfy their performance goals in the face of dynamically fluctuating power requirements.

5.5 Peak Load Provisioning

We next evaluate the use of dynamic knobs to reduce the number of machines required to meet peak load in



(a) swaptions

(b) x264

(c) bodytrack

(d) swish++

Figure 6: Power/QoS tradeoffs for each benchmark.

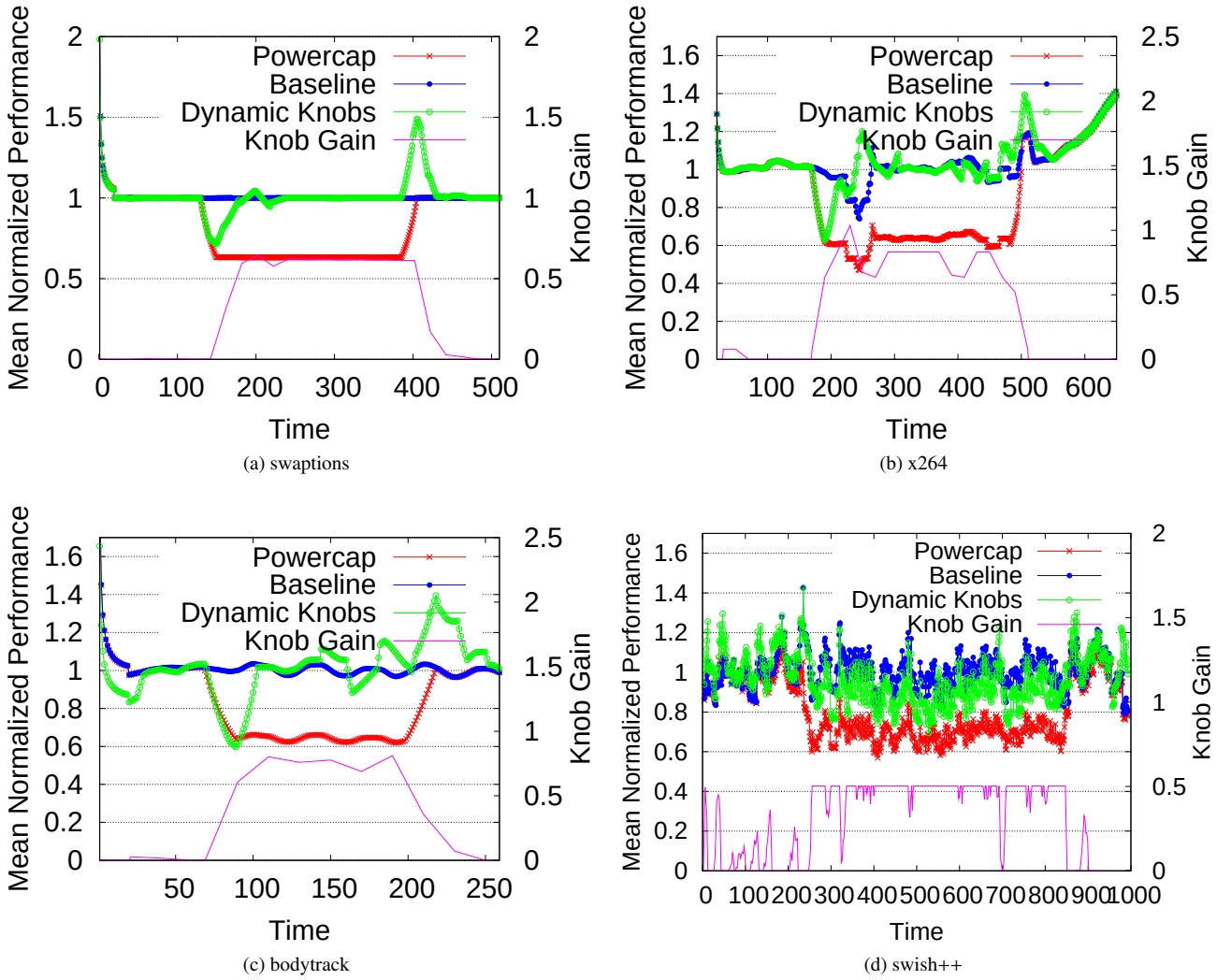
data centers, thereby reducing capital (e.g., machines), direct (e.g., power consumption) and indirect (e.g., cooling) costs.

We emulate data center peak load requirements by measuring the maximum per machine performance (see Equation 7). Given available machine resources, we limit the total number of machines to four. We next calculate the expected machine reduction using Equation 8. This value is determined by the maximum measured speedup achieved for a QoS bound of about 5% (except for swish++ where the bound is 30%).

For the three PARSEC benchmarks we provision for a peak load of 32 (4 machines · 8 cores/machine) concurrent instances of the application. The target performance for each instance is the same performance achieved running a single instance of the program on one core at the highest power state (2.4 GHz) and the default command line (which, for all applications, provides the highest QoS). Without dynamic knobs, these applications therefore require four machines to meet peak load. For swish++ we provision for a peak load of three concurrent instances, each with eight threads. The target performance is the maximum number of requests per second which can be served at the highest power state using the default command line. Thus swish++ requires three machines to meet this peak load.

Having determined the peak load and performance requirement, we then run each benchmark at varying levels of utilization and measure the system power consumption using the default command line with no elastic response. During this phase 1) the system load balances all jobs proportionately across available machines, and 2) we leave machines not assigned jobs idle but not powered off. These assumptions are based on a description of the usage patterns of Google servers [9].

We next compare the power consumption of the original system with that of a consolidated system which uses dynamic knobs to service the same peak load with fewer



(a) swaptions

(b) x264

(c) bodytrack

(d) swish++

Figure 7: Behavior of benchmarks with dynamic knobs in response to power cap.

machines. We reduce the number of machines in each system, enable the PowerDial runtime system, set the desired performance to the previously determined target, launch the same workloads used previously, and measure the power of this consolidated system. For low utilization the applications will run at the highest QoS level. As the utilization increases, the runtime will progressively manipulate the dynamic knobs to maintain the desired performance at the cost of some QoS loss. We provision the consolidated system based on the maximum speedup that dynamic knobs can deliver. For the three PARSEC benchmarks, we provision a single machine (to serve the peak load for the original four-machine system); for swish++ we provision two machines (to serve the peak load for the original three-machine system).

Figures 8a–8d presents the results of these experiments. Each graph plots the mean power consumption of the original (blue line) and consolidated (green line) systems (left y axis) and the mean QoS loss (red line, right y axis) as a function of system utilization (measured with respect to the original, fully provisioned system). These graphs show that using dynamic knobs to consolidate machines can provide considerable power savings across a range of system utilization. For each of the PARSEC benchmarks, at a system utilization of 25% consolidation can provide an average power savings of approximately 400 Watts, a reduction of 66%. For swish at 20% utilization, we see a power savings of approximately 125 Watts, a reduction of 25%. These power savings come from the elimination of machines that would be idle in the originally provisioned system at these utilization levels.

Of course, it is not surprising that reducing the number of machines reduces power consumption. A key benefit of the dynamic knob elastic response mechanism is that even with the reduction in computational capacity, it enables the system to maintain the same performance at peak load while consuming dramatically less power. For the PARSEC benchmarks at a system utilization of

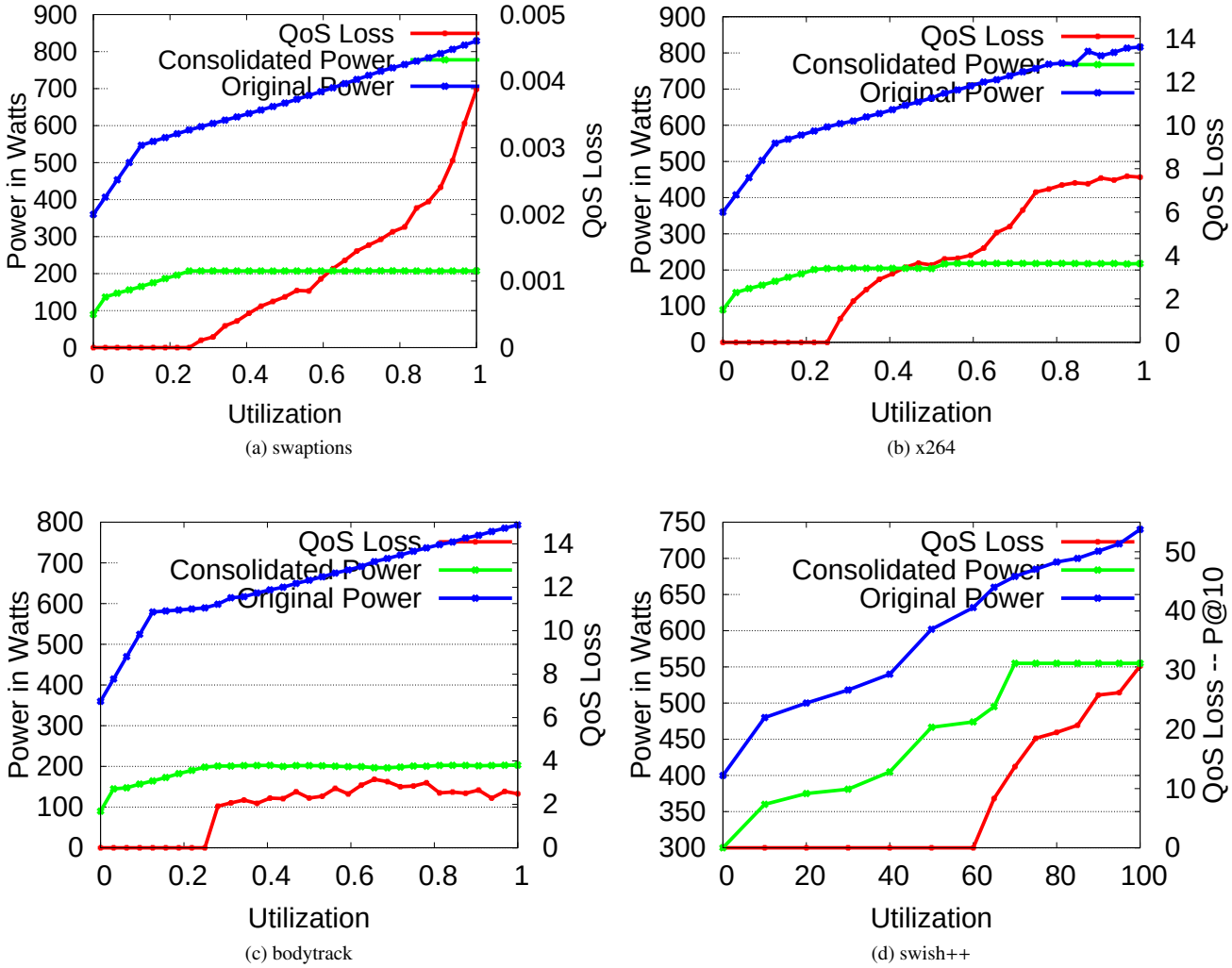


Figure 8: Using dynamic knobs to consolidate data centers and clusters to save power and money.

100%, the consolidated systems consume approximately 75% less power than the original system while providing the same performance. For swish++ at 100% utilization, the consolidated system consumes 25% less power.

The consolidated systems save power by automatically reducing QoS to maintain performance. For swaptions, the maximum QoS loss required to meet peak load is 0.004%, for x264 it is 7.6%, and for bodytrack it is 2.5%. For swish++ the QoS loss is 8% at a system utilization of 65%, rising to 30% at a system utilization of 100%. We note, however, that the majority of the QoS loss for swish++ is due to a reduction in recall; top results are generally preserved in order but fewer total results are returned. Precision is not affected by the change in dynamic knob unless the P@N is less than the current knob setting. As the lowest knob setting used by PowerDial is five, precision is always perfect for the top 5 results.

For common usage patterns characterized by predominantly low utilization punctuated by occasional high-utilization spikes [9], these results show that dynamic knobs can substantially reduce overall system cost, deliver the highest (or close to highest) QoS in predominant operating conditions, and preserve performance and acceptable QoS even when the system experiences occasional load spikes. Note that system designers can use the equations in Section 3 to choose a consolidation appropriate for their envisioned usage pattern that minimizes costs yet still delivers acceptable QoS even under the maximum anticipated load spike.

6 Related Work

Trading accuracy of computation for other benefits is a well-known technique. It has been shown that one can trade off accuracy for performance [28, 29, 15], robustness [28], energy consumption [15, 12, 31, 28] and fault tolerance [12, 31, 28].

Autotuners explore a range of equally accurate imple-

mentation alternatives to find the alternative or combination of alternatives that deliver the best performance on the current computational platform [35, 37, 16]. Researchers have also developed APIs that an application can use to expose variables for external control (by, for example, the operating system) [27, 6, 33]. This paper presents a system (PowerDial) that transforms static configuration parameters into dynamic knobs and contains a control system that uses the dynamic knobs to maintain performance in the face of load fluctuations, power fluctuations, or any other event that may impair the ability of the application to successfully service its load with the given computational resources. It also presents experimental results that demonstrate the effectiveness of its approach in enabling server consolidation and effective execution through power reductions (imposed, for example, by power caps).

Researchers have developed several systems that allow programmers to provide multiple implementations for a given piece of functionality, with different implementations occupying different points in the performance versus accuracy trade-off space. Such systems include Petabricks [7], Green [8], and Eon [30]. Petabricks is a parallel language and compiler that developers can use to provide alternate implementations of a given piece of functionality. Green also provides constructs that developers can use to specify alternate implementations. The alternatives typically exhibit different performance and QoS characteristics. PetaBricks and Green both contain algorithms that explore the trade-off space to find points with desirable performance and QoS characteristics. Eon [30] is a coordination language for power-aware computing that enables developers to adapt their algorithms to different energy contexts. In a similar vein, energy-aware adaptation for mobile applications [15], adapts to changing system demands by dynamically adjusting application input quality. For example, to save energy the system may switch to a lower quality video input to reduce the computation of the video decoder.

While each of these systems require source code modifications and/or annotations, this paper presents an approach that works directly on unmodified and unannotated applications. In this paper, we present a system that automatically transforms existing command-line configuration parameters into dynamic knobs, a control and actuation system that manipulates dynamic knobs with provably good convergence and predictability properties, and experimental results that demonstrate how dynamic knobs enable server consolidation and successful operation in the face of load and power fluctuations (induced, for example, by power caps).

We have also explored the use of loop perforation (which automatically transforms loops to skip loop iterations) to augment applications with the ability to operate at different points in an induced performance versus quality of service trade-off space [26, 21]. We have demonstrated that loop perforation can help developers find computations that are suitable for further optimization [26] and enables applications to adapt to fluctuations in the delivered computational resources [21]. This paper presents a system that uses dynamic knobs instead of loop perforation and more fully demonstrates how to use dynamic knobs to solve power management issues.

7 Conclusion

Power management is an increasingly important concern across a range of computing environments. Applications that can flexibly adjust their computing resource requirements, in combination with control systems that dynamically match their requirements to the fluctuating characteristics of the underlying computing platform, enable productive new solutions to many existing and future power management problems.

Our presented PowerDial system augments applications with dynamic knobs that the PowerDial control system can use to adapt the behavior of the application to execute successfully in the face of load spikes, power fluctuations, or (in general) any event that changes the balance between the computational demand and the resources available to meet that demand. Our experimental results demonstrate how PowerDial makes it possible to consolidate data centers (reducing operational and capital costs) and enables applications to operate successfully in the face of load and power fluctuations. We see PowerDial as an early example of an emerging class of management systems that will enable applications operate successfully in complex modern computing environments, which are increasingly characterized by fluctuations in power, load, and other key operating characteristics.

References

[1] Hp power capping and hp dynamic power capping for proliant servers. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01549455/c01549455.pdf.

[2] Intel Xeon Processor. http://www.intel.com/technology/Xeon.

[3] Project Gutenberg.

[4] Wattsup .net meter. http://www.wattsupmeters.com/.

[5] Xiph.org.

[6] Exposing application alternatives. In *ICDCS '99: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, page 384, Washington, DC, USA, 1999. IEEE Computer Society.

[7] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. Petabricks: A language and compiler for algorithmic choice. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Dublin, Ireland, Jun 2009.

[8] W. Baek and T. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2010.

[9] L. Barroso and U. Holzle. The case for energy-proportional computing. *COMPUTER-IEEE COMPUTER SOCIETY-*, 40(12):33, 2007.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT-2008: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, Oct 2008.

[11] M. Carbin and M. Rinard. Automatically Identifying Critical Input Regions and Code in Applications. In *Proceedings of the International Symposium on Software Testing and Analysis*, 2010.

[12] L. Chakrapani, K. Muntimadugu, A. Lingamneni, J. George, and K. Palem. Highly energy and performance efficient embedded computing through approximately correct arithmetic: A mathematical foundation and preliminary experimental validation. In *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, 2008.

[13] J. Deutscher and I. Reid. Articulated body motion capture by stochastic search. *International Journal of Computer Vision*, 61(2):185–205, 2005.

[14] X. Fan, W. Weber, and L. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture*, page 23. ACM, 2007.

[15] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, page 63. ACM, 1999.

[16] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, volume 3, pages 1381–1384. IEEE, 1998.

[17] B. Furht, J. Greenberg, and R. Westwater. *Motion Estimation Algorithms for Video Compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[18] V. Ganesh, T. Leek, and M. Rinard. Taint-based directed whitebox fuzzing. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 474–484. IEEE Computer Society, 2009.

[19] H.264 reference implementation. http://iphome.hhi.de/suehring/tml/download/.

[20] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments. In *ICAC'10: 7th International Conference on Autonomic Computing*, 2010.

[21] H. Hoffmann, S. Misailovic, S. Sidiroglou, A. Agarwal, and M. Rinard. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures . Technical Report MIT-CSAIL-TR-2009-042, MIT, Sept. 2009.

[22] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[23] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Controlling software applications within the heartbeats frame work. In *49th IEEE Conference on Decision and Control*, 2010. Under Review.

[24] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. Performance measures for information extraction. In *Broadcast News Workshop'99 Proceedings*, page 249. Morgan Kaufmann Pub, 1999.

[25] C. Middleton and R. Baeza-Yates. A comparison of open source search engines, 2007.

[26] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. ICSE, 2010.

[27] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: adaptive control of distributed applications. In *High Performance Distributed Computing*, Jul 1998.

[28] M. Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 324–334. ACM New York, NY, USA, 2006.

[29] M. Rinard. Using early phase termination to eliminate load imbalancess at barrier synchronization points. Oct. 2007.

[30] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: a language and runtime system for perpetual systems. In *SenSys '07*.

[31] P. Stanley-Marbell, D. Dolech, A. Eindhoven, and D. Marculescu. Deviation-Tolerant Computation in Concurrent Failure-Prone Hardware. 2008.

[32] SWISH++. http://swishplusplus.sourceforge.net/.

[33] C. Tapus, I. Chung, and J. Hollingsworth. Active harmony: Towards automated performance tuning. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 44–44, 2002.

[34] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. *Mobile Computing*, pages 449–471, 1996.

[35] R. Whaley and J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–27. IEE Computer Society, 1998.

[36] x264. http://www.videolan.org/x264.html.

[37] J. Xiong, J. Johnson, R. W. Johnson, and D. Padua. SPL: A language and compiler for DSP algorithms. In *Programming Languages Design and Implementation (PLDI)*, pages 298–308, 2001.